Objectgeoriënteerd programmeren: design patterns

Objectgeoriënteerd programmeren: design patterns

UML, Java, C# en PHP

Gabriel Sánchez Cano

Brinkman Uitgeverij · Amsterdam · 2019

ISBN 978 90 5752 398 4 / NUR 124

Omslagontwerp: Proforma, Barcelona
Redactie en opmaak: Henk Pel

Correspondentie inzake overneming of reproductie richten aan:
Brinkman Uitgeverij, Postbus 59686, 1040 LD Amsterdam
www.brinkman-uitgeverij.nl
tel. 020-4120970, fax 020-4120972
e-mail: info@brinkman-uitgeverij.nl

Inhoud

Voorwoord

Opbouw

Het boek bestaat uit vier hoofdstukken:

1 Inleiding UML
2 Objectgeoriënteerd programmeren in Java
3 Objectgeoriënteerd programmeren in C#
4 Objectgeoriënteerd programmeren in PHP

In **hoofdstuk 1** worden de basisconcepten van UML en UML-schema's behandeld.
Hoofdstuk 2 behandelt objectgeoriënteerd programmeren in Java en objectgeoriënteerde patterns (patronen).
Hoofdstuk 3 behandelt objectgeoriënteerd programmeren in C# en objectgeoriënteerde patterns (patronen).
Hoofdstuk 4 behandelt objectgeoriënteerd programmeren in PHP en objectgeoriënteerde patterns (patronen).
Bij de **eindopdracht** aan het einde van het boek, **hoofdstuk 5**, moet een applicatie ontwikkeld worden met gebruik van de MVC-methodiek. Dit mag ontwikkeld worden met eigen patterns in de programmeertaal naar keuze of met behulp van een framework.

Bij de opbouw is gebruikgemaakt van de taxonomie van Romiszowski waar onderscheid wordt gemaakt tussen kennis (het opslaan van informatie) en vaardigheden (acties uitvoeren om een doel te bereiken).

De meeste lesblokken eindigen met een kennistoets en een vaardigheid-lab. Elk hoofdstuk eindigt met een zelfstudieproject.

- **Kennistoetsen**: kennis van begrippen en procedures.
- **Vaardigheid-labs**: reproductieve vaardigheid, acties uitvoeren om een doel te bereiken.
- **Zelfstudieprojecten**: productieve vaardigheid. In tegenstelling tot reproductieve vaardigheden doen productieve vaardigheden een beroep op de creativiteit en planningsvaardigheden van de student; ze gaan gepaard met (complexe) beslissingsvorming op bewust of onderbewust niveau. De leerling moet de geleerde informatie spontaan toepassen in nieuwe situaties, waarin niet van tevoren geoefend is. Er moeten nieuwe oplossingen voor nieuwe problemen bedacht worden.

Dit boek is met de grootst mogelijke zorg geschreven. De juistheid en volledigheid van de gegevens kunnen echter niet worden gegarandeerd. De auteur en uitgever

aanvaarden geen aansprakelijkheid voor schade, van welke aard dan ook, die het directe of indirecte gevolg is van handelingen zoals onethisch hacken.

Ik wil al mijn studenten en collegae bedanken voor hun feedback tijdens het maken van dit boek. Ik heb dit boek met veel plezier geschreven en ik hoop dat zowel de studenten als docenten er met veel plezier mee zullen werken.

1 UML

1.1 Inleiding UML

Lesblok 1.1		
	Tijdsduur	3 uur
	Doel	Inleiding UML
	Benodigdheden	UML software-tool

Websoftwareontwikkeling

De volgende technieken kunnen gebruikt worden bij grote webprojecten:

- iteratieve ontwikkeling
- eisenanalyse
- gegevens modelleren
- coderen en testen

Daarnaast zijn er *good practice*-standaarden die zorgen voor de kwaliteit van het ontwikkelingsproces:

- codeer software die herbruikbaar is;
- codeer software die eenvoudig te onderhouden is;
- gebruik een ontwikkelingsplatform;
- documentatie aanleggen;
- houd logica, content en presentatie apart (Java, C#, JavaScript, PHP, HTML en CSS).

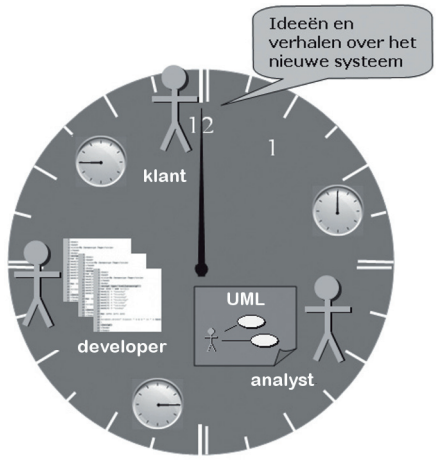
Websoftwareanalyse en -ontwerp

Softwareanalyse is een systematische benadering voor softwareontwikkeling. Dit is nodig voor complexe maar stabiele websites die goed te onderhouden zijn. Helaas ontbreekt softwareanalyse bij veel webprojecten.
Een reden is dat webontwikkeling vaak documentgeoriënteerd is: documentstructuur, grafisch ontwerp en productie. Deze benadering werkt prima voor kleine en statische sites, maar niet voor complexere sites.
Een tweede reden is dat scripting-talen, zoals HTML, JavaScript en PHP, gemakkelijk en toegankelijk zijn, zodat men meteen gaat coderen zonder veel aandacht voor analyse en beveiliging.

Ten slotte is er bij webprojecten vaak het idee dat er geen tijd is voor enige planning. De resultaten zijn onstabiele applicaties, gemiste deadlines en onleesbare scripts. Het ontwikkelen van webapplicaties is een nieuwe discipline waarbij een methodische benadering hard nodig is.

Iteratieve softwareontwikkeling

Iteratieve softwareontwikkeling is een systematische aanpak voor het plannen en bouwen van webprojecten. Iteratieve softwareontwikkeling zou je als volgt kunnen zien:

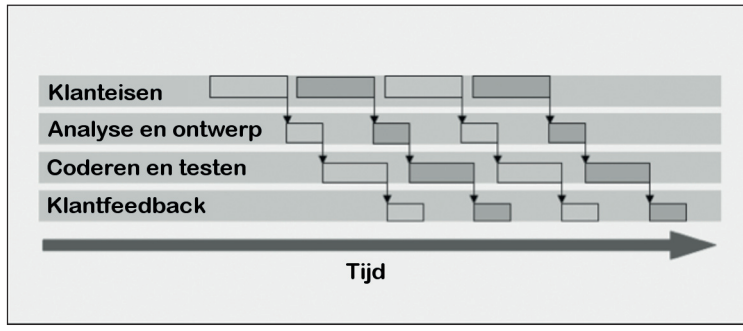


Figuur 1.1 Iteratieve softwareontwikkeling

Een iteratie is een cyclus, zoals de wijzers van de klok die ronddraaien. Eerst wordt de klant geïnterviewd. Vanuit zijn of haar verhalen en ideeën (user-stories) maak je een eisenanalyse. In dit hoofdstuk leren we vier UML-diagrammen kennen om informatiemodellen te maken. Deze modellen geef je aan de developer. De developer vertaalt je modellen naar programmacodes. Aan het einde van de eerste iteratie (cyclus) worden de resultaten gepresenteerd aan de klant. De klant geeft vervolgens zijn of haar mening en feedback. Deze feedback is het begin van de tweede iteratie.

Planning softwareontwikkeling

De planning van elke iteratie zou je als volgt kunnen zien:

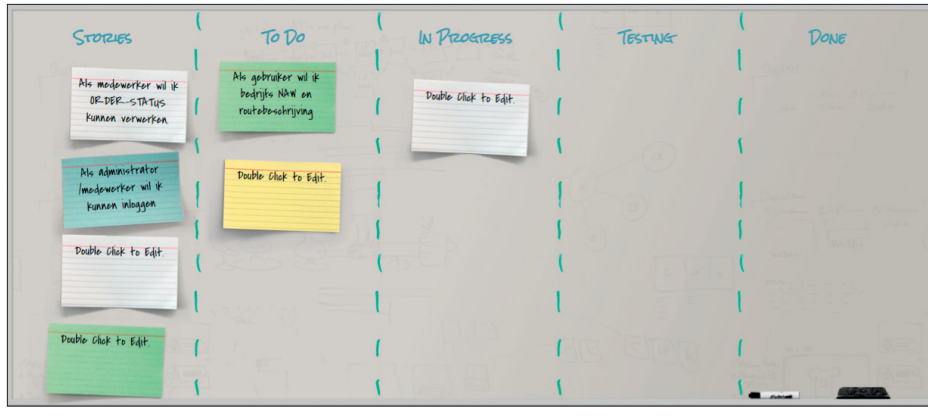


Figuur 1.2 Planning software-ontwikkeling

Vaardigheid-lab 1.1: Inleiding UML

Fasering en planning van tussenresultaten

Gebruik een software-ontwikkelmethodiek om de ontwikkelfases te plannen. SCRUM is bijvoorbeeld een eenvoudige maar effectieve methodiek waar je het programmeren van je user-stories mee kunt plannen.
Je vindt online een aantal SCRUM-boards zoals Trello.com. Hieronder zie je een voorbeeld van een SCRUM-board met user-stories.



Figuur 1.3 Planning software-ontwikkeling

Stories: zijn de verhalen (user-stories) uit interviews met gebruikers over de gewenste functionaliteit in het te bouwen system.
To Do: deze kolom is voor de user-stories die vandaag gestart worden.
In Progress: zijn de user-stories die nu gecodeerd worden.
Testing: deze kolom is voor de user-stories die nu getest worden.
Done: is de kolom voor de user-stories die nu gecodeerd en getest zijn.

SCRUM is een methodologie voor zelfsturende teams. De scrum-master is de facilitator die de processen en uitwisseling van informatie binnen het team beheert.

UML-diagrammen

Download en installeer gratis UML-tool-software voor het tekenen van UML-diagrammen zoals: Astah of StartUML.

Of maak kennis met online UML-tools zoals Lucidchart of Visual Paradigm.

Kennistoets 1.1: Inleiding UML

Bestudeer de theorie van dit lesblok en maak de volgende kennistoets.

Toets 1.1	
	1. Benoem drie technieken voor het ontwikkelen van webprojecten.
	2. Benoem drie best-practice-standaarden voor het ontwikkelingsproces.
	3. Wat zijn de fases van een iteratie van het ontwikkelingsproces?
	4. Wat doet de scrum-master?
	5. Wat is StartUML?

1.2 UML use-case-diagram

Lesblok 1.2		
	Tijdsduur	3 uur
	Doel	Use-case-diagram
	Benodigdheden	UML-software-tool

Unified Modeling Language (UML) is een modelleertaal die gebruikmaakt van grafische notatie om modellen te ontwerpen. UML bevordert de communicatie tussen workflowspecialisten, softwaredesigners en andere professionals, zoals mensen met kennis van een sector (bijvoorbeeld verzekeringen, gezondheidszorg of transport).

UML wordt gebruikt voor het maken van modellen voor alle mogelijke systemen:

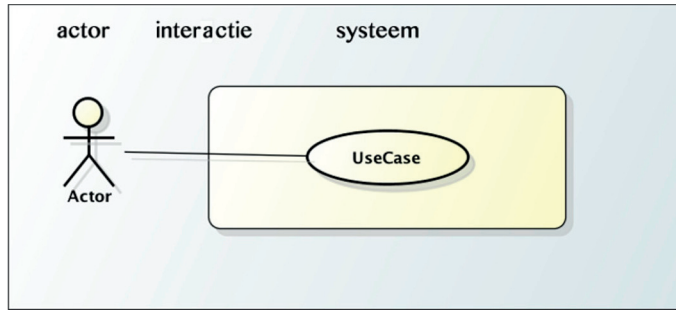
- complexe informatiesystemen;
- technische systemen zoals telecommunicatie;
- ingebedde realtime systemen zoals mobiele telefoons en auto's;
- softwaresystemen zoals besturingssystemen en databases;
- bedrijfssystemen.

Een model is geen definitieve oplossing. Het is maar een van de vele mogelijke oplossingen. Een bruikbaar model moet aan de volgende eisen voldoen:

- *accuraat*: beschrijft het te bouwen systeem;
- *consistent*: heeft geen conflicterende informatie;
- *begrijpelijk*: is eenvoudig uit te leggen en te wijzigen.

Eisenanalyse met use-case-diagrammen

De eisenanalyse volgt uit de wensen van de opdrachtgever. De ideeën en verhalen (user-stories) van de opdrachtgever vertaal je in een of meer use-case-diagrammen.
Een use-case-diagram beschrijft wat een systeem moet doen. Het bestaat uit meerdere use-cases die acties en interacties beschrijven tussen de actor (user) en het systeem. Een use-case-diagram gebruikt de volgende symbolen:

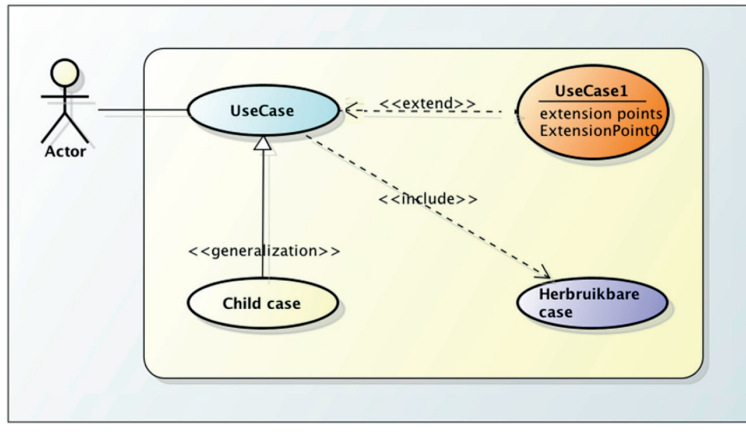


Figuur 1.4 Een actor in interactie met het systeem

Een actor mag een gebruiker of een ander systeem zijn. In de vorige figuur is UseCase de naam van de use-case.

Afspraken over notatie

Use-cases gebruiken de volgende notaties om scenario's te modelleren:



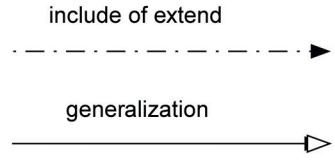
Figuur 1.5 Use-case-notatie

In figuur 1.5 zie je een actor in interactie met een systeem.

Je ziet ook verschillende relaties ontstaan tussen de use-cases.

- De actor komt in aanraking met een UseCase in het systeem.
- UseCase importeert een uitbreidingspunt (extension point) uit UseCase1.
- UseCase omvat (include) een Herbruikbare case. Dit is een afhankelijkheid (dependency).
- Child case is een generalisatie afgeleid uit UseCase.

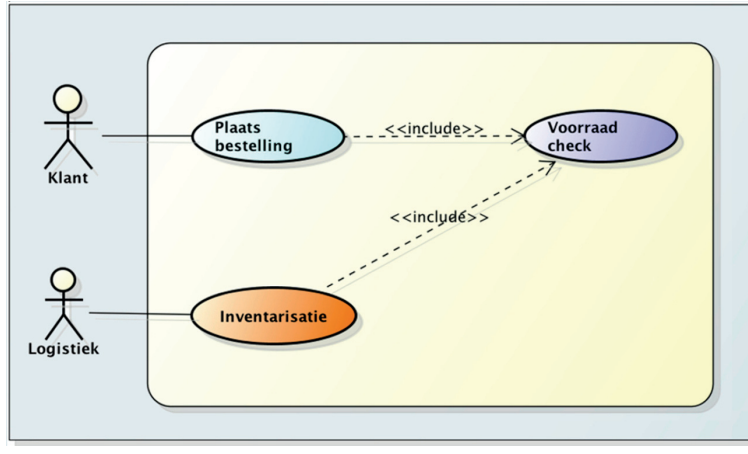
Een `<<include>>`- (omvatting) en `<<extend>>`-relatie (uitbreiding) geef je aan met een gestippelde lijn en pijl. Een `<<generalization>>`-relatie (generalisatie) geef je aan met een doorgetrokken lijn en pijl:



Figuur 1.6

De <<include>>-relatie

Gebruik `<<include>>` bij herbruikbare use-cases. Bijvoorbeeld, zowel de bestelling- als de inventarisatie-use-cases kunnen gebruikmaken van de voorraad-check-use-case. De voorraad-check-use-case mag gebruikt worden door andere use-cases.



Figuur 1.7 Een herbruikbare use-case

In dit scenario plaatst de klant een bestelling. De bestelling-use-case leidt tot een voorraad-check. De inventarisatie-use-case maakt ook gebruik van de voorraad-check-use-case.

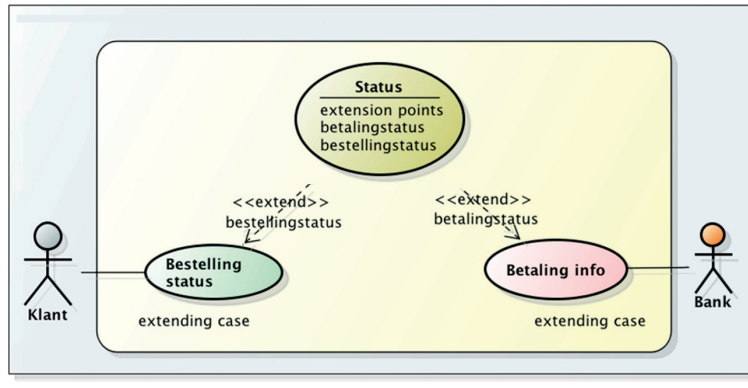
Wanneer je dezelfde functionaliteit identificeert binnen twee of meer use-cases, maak je hiervan een aparte single use-case. Zo kunnen meerdere use-cases dezelf-

de functionaliteit hergebruiken. Een use-case die een tweede use-case omvat is hiervan afhankelijk. Bijvoorbeeld, `Inventarisatie` is afhankelijk van `Voorraad-check` voor het uitvoeren van een inventarisatie. Een andere naam voor `<<include>>` (omvatten) is `<<dependency>>` (afhankelijkheid).

De <<extend>>-relatie

Een use-case kan extension-points hebben. Een extension-point is uitgebreide functionaliteit die toegevoegd kan worden aan andere use-cases. Een use-case die gebruikmaakt van een extension-point noemen we de extending use-case. Een extension-point is functionaliteit die je soms gebruikt. Dit wordt bepaald door de omstandigheden. Het verschil tussen een extend- en een include-relatie is dat een include-relatie altijd wordt uitgevoerd.

Neem bijvoorbeeld de use-case `Ambulance` die de use-case `Sirene` omvat. Dan zeggen we dat de ambulance een sirene heeft gekregen. Vervolgens hebben we de use-case `Geluiden` met de extension-points `politieGeluid` en `brandweerGeluid`. Dan kunnen we desgewenst aan onze use-case `Ambulance` de functionaliteit van het extension-point `politieGeluid` toevoegen.

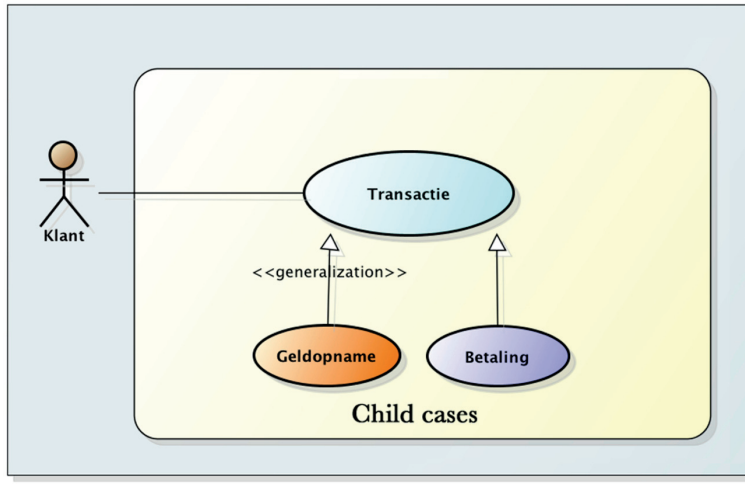


Figuur 1.8 Extending cases

Wanneer een use-case te complex wordt splitsen we de use-case op in extension-points. Een extending case kan een of meer extension-points uit een complexe use-case aanroepen.
In dit voorbeeld voert de extending case `Bestellingstatus` niet de volledige case `Status` uit, maar vraagt het specifieke extension-point `bestellingstatus` aan de use-case `Status`. We zeggen dat de extending case wordt uitgebreid. De pijl verwijst altijd naar de extending case.

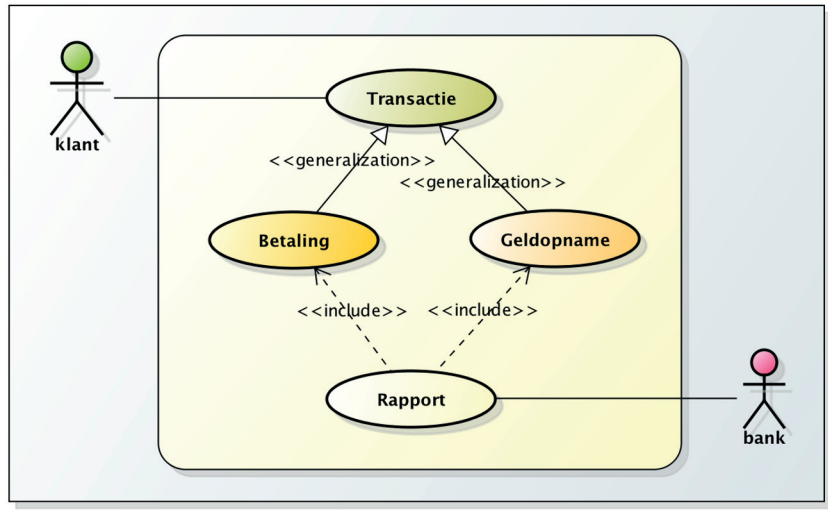
De <<generalization>>-relatie

Gebruik `<<generalization>>` bij een use-case die gelijksoortig is aan een andere use-case, maar gespecialiseerd is.



Figuur 1.9 Een generalization use-case

In dit voorbeeld is `Transactie` de parent-use-case en `Geldopname` en `Betaling` zijn de child-cases. De child-case lijkt erg op de parent-case, maar is wel een speciale versie van de parent-case. Deze relatie noemen we ook een 'is een'-relatie. Bijvoorbeeld, een betaling **is een** transactie. De pijl wijst altijd naar de parent case. In de volgende figuur zien we een voorbeeld van de generalization-relatie:



Figuur 1.10 Generalization met include

Het diagram beschrijft het volgende scenario:

- Klant voert een transactie uit.
- Het systeem biedt twee opties aan: `Geldopname` of `Betaling`.
- De bank stelt een rapport op dat afhankelijk is van de transacties.